Solist or How To Look For a Needle in a Haystack? A Lightweight Multi-Overlay Structure for Wireless Sensor Networks

Yann Busnel, Marin Bertier, Anne-Marie Kermarrec

SOLIST or How To Look For a Needle in a Haystack? A Lightweight Multi-Overlay Structure for Wireless Sensor Networks

Yann Busnel
IRISA / Univ. Rennes 1
Campus universitaire de Beaulieu
35042 Rennes Cedex
Email: Yann.Busnel@irisa.fr
Tel: +33 2 99 84 75 52

Marin Bertier
IRISA / INSA Rennes
Campus universitaire de Beaulieu
35042 Rennes Cedex
Email: Marin.Bertier@irisa.fr
Tel: +33 2 99 84 22 06

Anne-Marie Kermarrec
INRIA Rennes - Bretagne Atlantique
Campus universitaire de Beaulieu
35042 Rennes Cedex
Email: Anne-Marie.Kermarrec@inria.fr
Tel: +33 2 99 84 25 98

***Abstract*—In this paper, we consider sensor database systems. Sensors are attached to objects and queries on the objects are operated at the sensor network level. Although queries to such a system might be extremely complex, ensuring efficiently basic functionalities such as broadcast or anycast without any central element is not trivial. In this paper, we provide a suite of *-cast (anycast, k-cast, broadcast) functionalities in a fully decentralized manner. More specifically, we present the design and evaluation of SOLIST, a multi-layer structure for sensors, largely inspired from structured peer-to-peer systems providing such functionalities. The main goal of SOLIST is to limit the overall energy consumption. A type is associated to each sensor, and the *-cast functionalities are implemented at a type granularity regardless of the number of types and their distribution within the network. A typical use of such a system is sensor-based stock management. We evaluate SOLIST through simulations and show that SOLIST achieves a reasonable trade-off between performance and energy consumption.**

I. INTRODUCTION

Looking for a specific information in a large-scale network with no underlying structure is as difficult as searching a needle in a haystack without any metal detector. The most natural way to explore the network is flooding. Yet, this is not desirable in wireless sensor networks (WSN) for which energy saving is a first-class concern. WSNs are composed of a large number of small entities, with much less capabilities than common computers. Due to their tiny size, these entities, equipped with a wireless communication facility and called *sensors* in the following, possess slim resources in terms of memory, CPU, *etc*. [1], [13].

As opposed to general-purpose large-scale distributed systems, WSN are deployed and configured usually to fulfil a specific application needs. Example of such applications are monitoring, stock management, data aggregation, *etc*. Yet some basic functionalities are common to a large number of those applications. We consider applications in which sensors are associated to a specific type. A typical example is stock management. Each sensor belongs to a given type, which can dynamically change with respect to application needs or execution. At the heart of data management, we have identified a set of communication primitives that can be seen as basic building blocks for those applications. We call the **-cast suite*, this set of functionalities, which consists in: *anycast*, *broadcast* and *k-cast* queries that aim at reaching respectively one, all or k entities of a given type (where k is a given parameter of the k-cast primitive). To illustrate our purpose, consider a stock management application, where each sensor represents a physical item of a given type. Sending a message to all the sensors of a given type or querying the system to know whether there still exists an instance of a given type, are standard operations. The aforementioned *-cast suite would provide the means to implement easily such operations.

In this paper, we present the design and evaluation of SOLIST (*Self-Organized Large-scale and lightweight Information-based Sensor Technology*), a structured overlay network for WSN providing an efficient *-cast suite. Based on a simple common interface, SOLIST provides a generic infrastructure, relying on a lightweight multi-layer structure, while ensuring low energy consumption. Sensors are clustered according to their type into specific layers. In this paper, due to space constrains, we only present the anycast operation. Note that this operation can be used to provide an entry point to a specific type layer and therefore implement in those layers the k-cast and the broadcast [3].

The rest of the paper is organized as follows. In Section II, we briefly introduce the generic interface. The SOLIST design is presented in Section III. In order to evaluate SOLIST, we conducted simulations in worst-case scenarios and compared it against traditional approaches. We present the experimentation results in Section IV. Finally, Section V briefly surveys the related works before concluding in Section VI.

II. THE *-CAST SUITE

As mentioned before, a large majority of distributed applications relies on a few basic functionalities. More specifically, in WSNs, accessing application nodes according to their type is a common task in many applications. SOLIST relies on a group-based structure to provide a basic set of functionalities, identified as *-cast in the sequel. Nodes may be assigned to a given *type*. This type may be static (representing the shape of the sensor in case of heterogeneous network for instance) or dynamic (related to the level of energy available, the sensed data, *etc.*). Note that some nodes may not be directly assigned to a specific type, and thus contribute only to the global connectivity of the network. SOLIST ensures that every nodes of the same type are dynamically *clustered*

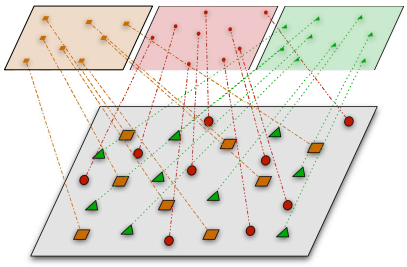

Fig. 1. Projection into layers

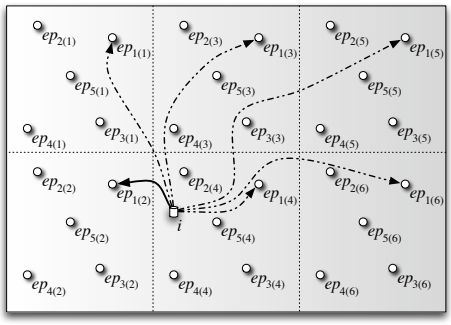

Fig. 2. Example of a 3×2 cell namespace divisions.
Node i sends its request for layer 1 to the nearest entry point: $ep_{1(2)}$

together. Our system allows to find efficiently a node from a given type, and to navigate easily between all nodes sharing the same type. SOLIST can be adapted to any application by adapting the matching between types and nodes. SOLIST provides the following set of core functionalities, common to many applications:

Anycast [ANYCAST($type$)] *This service aims at reaching one node, if any, of a specific $type$.*

For example, this functionality may be used to discover if one instance of a given type exists in a system, and to localize it.

k-cast [KCAST($type, k$)] *This service aims at reaching k nodes, which belong to a specific $type$, should they exist. This function returns* TRUE *if they are at least k nodes of the specific $type$,* FALSE *otherwise. Moreover, this operation provides a direct access to those k instances, although we could consider attaching more information into reply messages.*

For instance, in stock management applications, it is useful to know if entities of a specific type are available in sufficient quantity. In fire monitoring applications, such applications trigger an alert if say 10 sensors have sensed a temperature greater than 40 degrees, or a hygrometry value lower than 2 %vol. Furthermore, firemen may also be want to be informed of the average value of a random sample of nodes sharing the same type. k-cast might be used to perform these tasks.

Broadcast [BROADCAST($type$)] *This service aims at reaching all nodes belonging to a specific $type$.*

Using this functionality, SOLIST may reach all nodes belonging to a a specific type of nodes, this last service can be viewed as a *multicast* one. Such a functionality may be used either to disseminate information to every nodes of a given type, as in a publish-subscribe systems for example. It can also obviously be used to enumerate the number of nodes of a given type.

III. SOLIST CORE

A. A multi-layer structure

In order to provide the *-cast suite, SOLIST is built upon a multi-logical layer structure, clustering the network according to nodes' type. This allows navigating between nodes sharing the same type and provides an efficient localization based on *entry-points* (*cf.* Section III-C). Each node belongs to the common basic layer, used essentially to ensure the global connectivity of the network. To this end, each sensor is aware of its own virtual coordinate in a relative Cartesian space. Messages in the basic layer are routed using a lightweight geographic routing protocol. This enables to reach any destination based only on its virtual coordinates.

The multi-layer structure provides a logical clustering of the network. On top of this ground layer, SOLIST implements a specific lightweight overlay per group. For instance, in Figure 1, n nodes are spread between three different types. For each type corresponds a specific overlay (*i.e.* the triangle one, the circle one and the square one) on top of the ground layer. Communications between two nodes of a specific overlay are implemented using the ground layer.

B. LIGH-t-LAYER *structure*

Each of the aforementioned logical layers has the same framework, based on rectangle-shape space division. In this respect, WSNs are similar to peer-to-peer (P2P) overlays [2] (in term of properties and functionalities). Thus, the proposed structure is largely inspired from a classical P2P Distributed Hash Table (DHT), namely CAN [12]. A previous work on evaluation of P2P structured overlays for multicast [4] shows that CAN provides an ideal structure for our targeted functionalities in a WSN. In a nutshell, CAN splits the space into logical responsibility areas, evenly spread between nodes in the system. To maintain this structure, each node owns a virtual neighborhood, corresponding to the representatives of its own zone's adjacent areas. We slightly modify this skeleton to match the WSN setting. The main particularity of this context is that communication and energy constraints have to be considered in priority. We denote this revised structure by LIGH-t-LAYER in the following, where t corresponds to the type t shared by the nodes in this layer. Note that in our system, each node maintains the same virtual coordinates for any overlay it belongs to and the responsibility areas are only used for k-casting and broadcasting efficiently in a group.

Let us introduced how this network is gradually built. Initially, the first joining node of a given type t becomes *responsible* of the whole LIGH-t-LAYER area. When another node joins the network, it contacts the representative of the area responsible of its coordinates. This area is split between

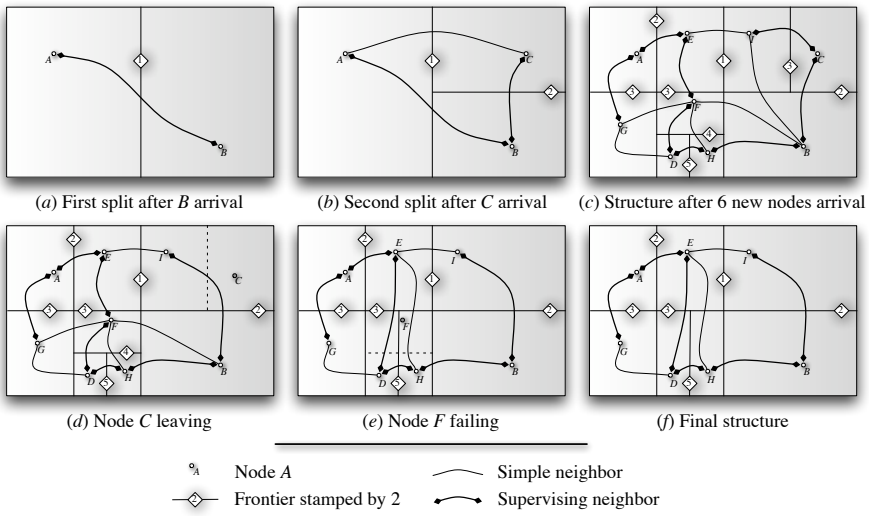

Fig. 3. Evolution of a LIGH-t-LAYER structure following different kinds of event

the two nodes (as it is done in CAN). Thus, a new frontier between these two new, smaller, areas is created and stamped in order to maintain the historic of frontier's creations. We develop below the role of these stamps to cope with nodes departures or failures. Each node belonging to a specific layer has to maintain a list of LIGH-t-LAYER neighbors, denoted $view_t$. Each $view_t$ entry contains (1) the neighbor's id; (2) the neighbor's coordinates; (3) the origin and final coordinates of the shared frontier and (4) the frontier stamp.

In order to illustrate a LIGH-t-LAYER structure evolution, Figure 3 presents different events from the start. First, node A joins the network and becomes responsible of the whole logical area. Figure 3.a shows the state of the layer after B's integration: it initially contacts A, which splits its own area in two and then, returns the coordinates of *(i)* the B's area and *(ii)* the created frontier between them. Later, at C's arrival, B splits its own area and stamps the new frontier by a strictly greater identifier as shown in Figure 3.b. Finally, Figure 3.c presents the layer structure state incorporating 9 nodes, labelled from A to I. In that case, we observe that the largest frontier stamp is equal to 5.

The bottom part of Figure 3 presents how to keep a consistent structure in case of a node departure (*cf.* Figure 3.d) or a failure (*cf.* Figure 3.e). Departures and failures are handled the same way in LIGH-t-LAYERs. When a node chooses to leave the overlay, it sends its $view_t$ to the representatives across the last created frontier. These last are potentially several, as across the tag-4 frontier in Figure 3.c for instance. Thus, these nodes get the responsibility of an extended area, update their $view_t$ according potential new neighbors, stretch out their frontiers and finally, send updated information to the concerning neighbors. For instance, node C, in Figure 3.d, has to leave the layer. C sends its $view_t$ to node I across its tag-3 frontier (here, for node C, 3 is the highest frontier stamp). Then, I becomes the representative of the merged area: C's union I's one. Finally, I updates its $view_t$ according to the new extended frontier with B's area, and sends the new coordinate to B.

In parallel, in case of failure, as presented in Figure 3.e, node F is removed from SOLIST without warning. F can obviously not send its own $view_t$ to any neighbor, as in the leaving procedure. Also, we integrate a supervision mechanism in our structure. Each node, which belongs to an overlay, periodically check upon one or two of its neighbors to prevent failures. This supervision links are represented by bold arrays in Figure 3. Thereby, node D, which is the F's supervisor, sends a *collectView* request to learn *a posteriori* the F's $view_t$. This request is routed all around the past F's area and finally, comes back to D. Using this information, D falls trivially into a classical leaving procedure: it contacts all nodes across the deleted frontier (here, node H besides itself) and merges the corresponding F's sub-area with its own one. Figure 3.f presents the state of this LIGH-t-LAYER structure once those actions have been performed.

However, in order to take advantage of this overlay structure, we require an efficient and robust way of reaching a specific LIGH-t-LAYER from anywhere in the network. This is the goal of the following subsection.

C. Linking the world: Entry points

A challenging issue remains to reach a specific LIGH-t-LAYER from any node, without any other information than the type t identifier. To this end, we map a logical namespace on top of the ground layer, inspired by recent works [11]. In addition, we use two mutual hash functions in order that each node shares a set of common coordinates in this namespace, computed using these functions. The aim of these hash functions is to uniformly spread these coordinates in the space, based on the number of types. In the following, we denote these coordinates as *entry points*.

Let t correspond to the type identifier of the searched LIGH-t-LAYER. Let $f_x : \mathbb{N} \rightarrow [0;1[$ and $f_y : \mathbb{N} \rightarrow [0;1[$ be two hash functions and wsX, wsY the respectively horizontally and vertically size of the relative Cartesian space. Entry point coordinates corresponding to this layer (denoted by ep_t) are computed as follows:

$$ep_t = (x, y) \text{ where } \left\{ \begin{array}{rcl} x & = & f_x(t) \times wsX \\ y & = & f_y(t) \times wsY \end{array} \right. \quad (1)$$

As f_x and f_y are shared by all nodes, entry point coordinates in the network are unique for a given type t. However, as an entry point is represented by its uniformly random generated coordinates, there might not be any node at this specific location. Therefore, we consider no distinction between a point in the virtual space and the nearest node of this point. Both are called *entry point* in the sequel. This nearest node of ep_t has to be acquainted with at least one node of type t (and so, to be aware of the existence of the corresponding LIGH-t-LAYER). If this LIGH-t-LAYER exists, it is able to reply with the identifier and the coordinates of a node belonging to this layer. In the following, this node replied by an entry point is denoted a *contact node*.

In order to balance the load of the nearest node to an entry point and to avoid that some requests have to traverse the whole network to reach an entry point, the namespace is divided according to a $m \times n$ grid, on x and y coordinates. Then, the namespace is mapped onto each division, which are called *cells* in the following. When a node wants to access a specific layer, it sends its request to the nearest entry point of the given type. For instance, Figure 2 presents a $m \times n = 3 \times 2$ cell topology with 5 identified types. For each cell, 5 entry points (et_1 to et_5) have the same relative position. Here, node i sends a request to access the layer associated to type 1. i is aware of all the 6 entry point location (dash and solid arrows) and sends its request to the nearest one (here, $ep_{1(2)}$ linked by a solid arrow). Let $d(\cdot,\cdot) : (\mathbb{R} \times \mathbb{R})^2 \mapsto \mathbb{R}$ the Euclidian distance between two points in the virtual space and csX, csY the cell size respectively horizontally and vertically. To find the nearest entry point, node i has to compute the Equation 2 calculus. Let the two sets $\Gamma_{x,t} = \{(k_x + f_x(t)) \times csX | k_x \in [0..m-1]\}$ and $\Gamma_{y,t} = \{(k_y + f_y(t)) \times csY | k_y \in [0..n-1]\}$

$$ep_t = (x, y) \text{ st. } d(i,(x,y)) = \min_{\substack{x' \in \Gamma_{x,t} \\ y' \in \Gamma_{y,t}}} d(i,(x',y')) \quad (2)$$

Last but not least, considering this searching layer mechanism, in order to update the entry points' knowledge, when a node joins a specific LIGH-t-LAYER, it sends its arrival information to the nearest type t entry point. Therefore, this node is always informed of the nearest node belonging to the given LIGH-t-LAYER layer. Thus, it can send information up-to-date about a close node when a "search layer request arrives". Likewise, in case of failure or departure, entry points are informed too in order to update as well their contact node links. As entry points choose locally their nearest contact node, the leaving/faulty node is not aware of the entry point referenced. So, all entry points have to be informed of each change in the network. This is precisely this mechanism, which is used to reply to anycast requests.

IV. EVALUATIONS

a) Simulation environment: We evaluate SOLIST with SeNSim, a discret event simulator developed by the ASAP/IRISA project team [14]. SeNSim allows analysing SOLIST under different topologies, and with different failure and stimulus scenarios.

We consider a 1,000 sensor network, sensors are spread uniformly in a square area of 96×96 meters. The transmission range of a sensor is 0,7 meter. The repartition of sensors into 10 types is static (10 nodes of type 1, 30 of type 2, 50 of type 3, ..., 190 of type 10),

In each evaluation, if not specified otherwise, each node joins the network, launches a broadcast and a k-cast[1], and finally leaves the network. Arrival and departure dates are randomly generated, as well as the requested type for broadcast and k-cast. The k value is picked at random between 1 and 200, in order to get some `true` and `false` replies.

b) Routing protocol: SOLIST structures the network to localize a given sensor type in the network. From SOLIST, a sensor obtains the coordinate of the request destination. To be able to reach the destination, we use a light GPSR geographic routing protocol [8]. We don't use the computation of *planar graph* required by the original algorithm because this computation requires non-negligible energy consumption and we want to focus the energy consumption measurements on SOLIST only. Therefore, we only use the greedy routing protocol with the *right-hand rules* in case of void density on a route.

c) Algorithms comparison: We compare the *-cast algorithms build on SOLIST against two existing protocols: (1) anycast against the simple but naive *Random Walk* mechanism to find one node of a specific type in an unstructured network and (2) *Flooding* for broadcast. Using (1), the node sends a request to one of its direct neighbors (*i.e.* in WSN context, one of the nodes in its transmission range). If this node belongs to the requested type, it sends to the initiator its position. Otherwise, it randomly chooses a node among its neighborhood (except the previous sender), and sends the request to it, and so on. After a predefined number of hops, if no node is found, the last request receiver replies a `NOT_FOUND` message to the initiator.

d) SOLIST*'s anycast:* We evaluate the SOLIST's anycast mechanism along two metrics:

- the distance between a node and the anycast destination;
- the distance between a node and its closer entry point.

Figure 4 presents the average distance between the anycast query initiator and the contact node with the requested type for several numbers of cells configuration. For comparison, we also represent the average distance between the initiator and the destination and the distance obtained with Random Walk.

[1] the algorithms of the *-cast suite. are presented in [3]

The contact node provided by SOLIST is the closest node known by the entry point contacted by the initiator. Consequently the lower the cell size, the nearer the contact node. A side effect can be observed for huge number of cells, where the SOLIST anycast plot slightly grows up. Effectively, when a node joins a layer, it informs the nearest corresponding entry point. So, as the cells size decreases, some entry point may be nearer to the new node, without being aware of its arrival (we avoid broadcasting all entry points at each arrival to reduce energy consumption).

Figure 5 presents a cumulative distribution function (CDF) of this distance for a 5×5 cells configuration in SOLIST against an average random walk approach. In these simulations, more than 95 % of nodes with SOLIST have a distance less than 13.5 meters, while only 80 % fulfil this criterion when the random walk approach is used. Furthermore, the tail of SOLIST plot shows that all nodes have a distance less than 62.5 meters while the random walk mechanism results in a maximum distance of 89, which is about the size of the simulation environment.

e) Entry point distribution: Another interesting metric to evaluate the SOLIST anycast mechanism is the distance between a node and the nearest entry point. Figure 6 presents the average value of this distance and the standard deviation according to the number of cells. The distance is represented as in previous figures with the Euclidian one, and also with the number of hops needed to reach the entry point. As predicted, the greater the number of cells, the smaller the cells size and the nearer the entry point. An interesting point is that from minimum 6 cells, the average number of hops is lower than 3 hops. In consequence, energy consumption spends to reach a contact node is low. Figure 7 presents a CDF of these two distances for a 5×5 configuration in SOLIST. 90 % of nodes can reach the nearest entry points in at most 1.5 hops[2], and 98.5 % by at most 2.5 hops. In this experiment, each cell has a 20×20 square meters size. No nodes are at a distance to an entry point of more than $\sqrt{2}\times20$ meters (the diameter size of a cell) as predicted, but some are at a distance between $\sqrt{2}\times20$ and $\frac{\sqrt{2}\times20}{2}$: these nodes are the ones, which are located on the border of the network. They have fewer choices for entry points than nodes located in the centre of the network.

f) SOLIST *energy consumption:* In order to evaluate and compare SOLIST, we have run several simulations with the same network behavior (as join date, leave date, failure date, number of events, *etc.*) Each simulation lasts 10,000 discrete times. Each node joins the network once and leaves or fails before the end of the simulation. Each node sends one broadcast and one k-cast request. As each join or *-cast request needs one anycast mechanism, these workloads consists in 3,000 anycast, 1,000 broadcast, 1,000 k-cast[1]. Moreover, join, leave and maintenance consumption have to be taken into account. To imitate real energy consumption, we use the power characteristics described in Table 8, proposed in [10] from MICA nodes measurement. We consider that each node has a full battery of 2,200 mAh at the beginning of the simulation.

Figure 9 presents for several cells configuration in SOLIST, the average total energy consumption at the end of the simulation against the one generated using random walks and flooding. This demonstrates the efficiency of SOLIST in saving energy for the whole network. The consumption in SOLIST is slightly increased according to the number of cells, due to the consumption needed for maintenance as we see below.

Figure 10 presents for several cells configurations in SOLIST and random walk, the average anycast energy consumption and Figure 12 and 13 present the end simulation energy snapshot respectively for SOLIST and Random Walk. The first one shows the interest of SOLIST by using at least 3 cells in this network topology. As tiny cells do not answer every time with the nearest contact point, the anycast mechanism increases slightly according to the cell size. Although, SOLIST anycast energy consumption required is no more than 35 % compare to the random walk mechanism. The two other figures present some compactness of higher consumption points. For SOLIST's anycast, these points correspond to the location of the 80 entry points in the system. We can easily infer the 8 cells from this figure. Contrarily, using the random walk mechanism, the point of higher consumption corresponds to the node with the highest density in the topology. This is due to the fact that requests have a higher probability to stay in this high-density neighborhood (random walks have a low probability of visiting low-density neighborhood). SOLIST is always lower than the ones observed using random walks.

Finally, average structure maintenance energy requirement is presented in Figure 11 according to number of cells in SOLIST. As join operations and reorganizations in case of failure are only a local task, the associated energy consumption remains low, strictly lower than 0.25 % of the total amount of energy available for each one. Leaving operations require more energy when the number of cells is growing. This observation is a consequence of the linear expansion of leaving messages to inform each entry points. For instance, in 10×10 cells configuration, as each node leaves or fails during the simulation, 100,000 leaving messages are generated among the network...

The results show that SOLIST outperforms the considered alternatives with respect to energy consumption. For the topology considered for this evaluation (*i.e.* 1,000 nodes among 10 types), an 8 cells configuration is ideal to obtain the best trade-off between efficiency and energy consumption.

V. RELATED WORKS

In this section, we present various works related to SOLIST. To the best of our knowledge, providing a common application-programming interface (API) for WSNs has not yet been proposed. Also, we are comparing the *-cast suite elements with previous works. These works address one of the several functionalities introduced in this paper.

[2]these results are given for a round trip message divided by 2 in order to take into account the non-symmetric route from a node to another with GPSR.

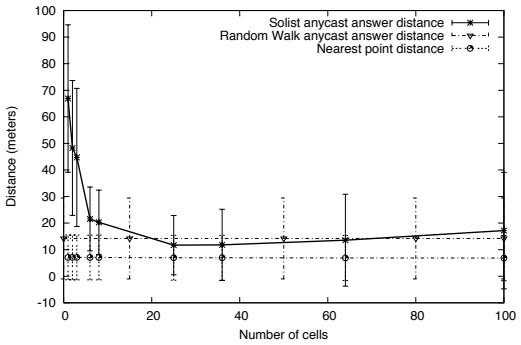

Fig. 4. Average distance of the contact node known by an anycast request.

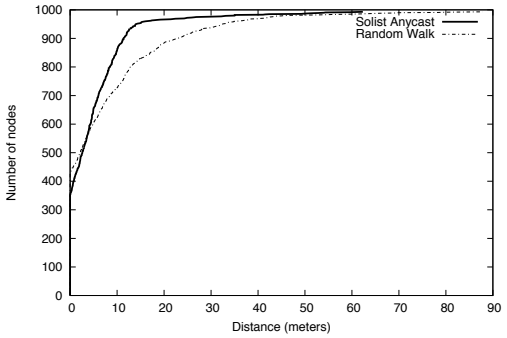

Fig. 5. CDF of Figure 4 for a 5×5 cells topology.

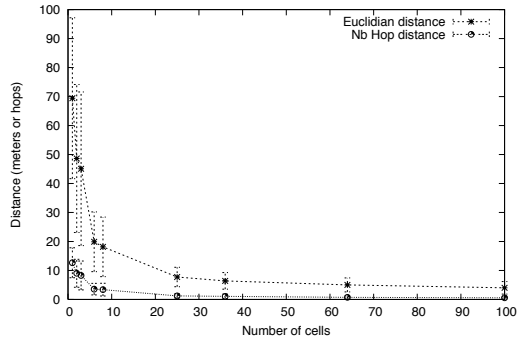

Fig. 6. Average distance between the request node and the entry point.

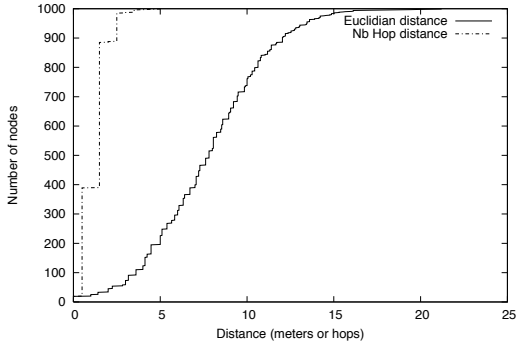

Fig. 7. CDF of Figure 6 for a 5×5 cells topology.

Operation	nAh
Transmitting a packet	20.000
Receiving a packet	8.000
Radio listening for 1 millisecond	1.250
Flash Read Data	1.111
Flash Write/Erase Data	83.333

Fig. 8. Typical power requirement for various operations of a Mica mote proposed in [10].

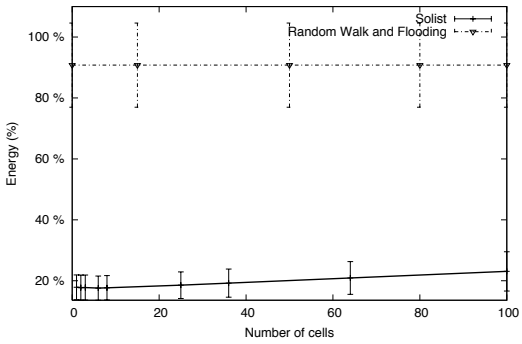

Fig. 9. Average total energy consumption.

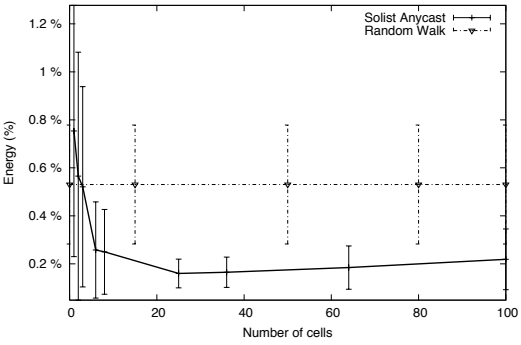

Fig. 10. Average anycast energy consumption.

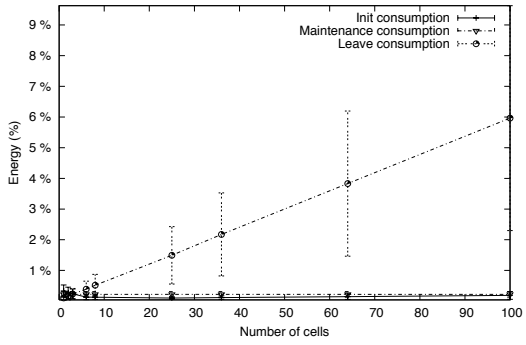

Fig. 11. Average structure maintenance energy consumption.

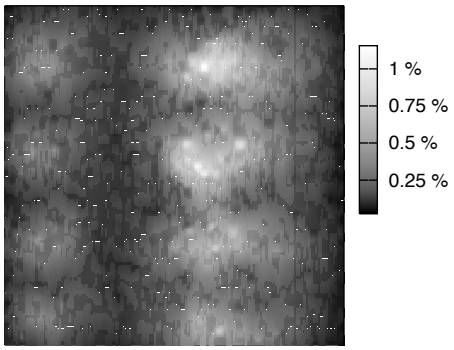

Fig. 12. Energy consumption at the end of the simulation for anycast in SOLIST with 2×4 cells.

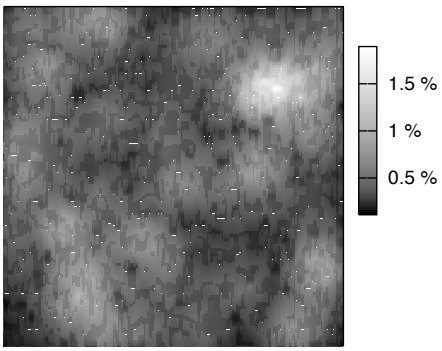

Fig. 13. Energy consumption at the end of the simulation for anycast with random walks.

On one hand, we consider previous work on anycast in WSNs. For instance, in [15], authors propose an anycast routing protocols based on hierarchical tree. These works used a base station in order to build the routing tree. Although this is relevant in some contexts, SOLIST doesn't required any particular node and allow each node to consult information about the network. On the other hand, we consider works on multicast in WSNs. This can obviously be compared to a broadcast for a specific type of sensors in SOLIST. Several works have been proposed recently: an interesting result is presented in [16]. This paper proposes a broadcast and a multicast mechanism for WSNs based on tree construction. A tree-based structure has to be built for each multicast group by deleting useless links from the broadcast tree. This system has been evaluated only on a small network. Moreover, the system does not consider node failure or departures. Another approach for multicasting in WSNs [5] considers only reliability for any suitable multicast protocol, by lost message recovering and a last one [7] considers multicast on mobile sensors, which is not our context. The nearest contribution has been proposed recently in [6]. Authors proposed a routing protocol for anycast and multicast in WSNs. They present a theoretical analysis of their scheme, based on dominance net construction. This paper presents interesting results but relies on a different model. Moreover, dynamic multicast group management requires the construction of a minimum spanning tree for every group modification. Finally, no simulation or experimentation has been done to illustrate the theoretical results. So, comparison is hard, as we do not provide such theoretical results.

Finally, we must cite a relevant approach to deal with information in a WSNs. TinyDB [9] is based on information acquisition directly on the network as us, but viewing the network as a physical database. Consequently, they propose a query language based on extension of SQL. As SOLIST, TinyDB is generic and represents an all-in-one solution for WSN application. TinyDB is application dependant according to query optimization and execution.

VI. CONCLUSION

In this paper, we propose an effective all-in-one solution for the *-cast suite (*i.e.* anycast, k-cast and multicast) in static WSNs. This contribution, called SOLIST, is a generic lightweight system architecture for large-scale WSNs. SOLIST is composed of a finite set of overlays (LIGH-t-LAYER) providing a common interface, with a type-based clustering. Associated with the proposed searching layer mechanism, SOLIST provides an efficient *-cast implementation in term of energy saving and reliability. We evaluate by simulation each functionality provided by SOLIST and we compare it against two other standard algorithms (flooding an random walk). Results demonstrate that SOLIST outperforms these algorithms in term of energy saving and therefore provides a good trade-off between functionality and energy consumption.

REFERENCES

[1] I. F. Akyildiz, W. Su, et al. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.
[2] Y. Busnel, M. Bertier, et al. GCP: Gossip-based Code Propagation for Large-scale Mobile WSNs. In *Proc. of Autonomics*, 2007.
[3] Y. Busnel, M. Bertier, et al. SOLIST: A Lightweight Multi-Overlay Structure for Wireless Sensor Networks. Research R. 6404, INRIA, Rennes, France, Oct. 2007.
[4] M. Castro, M. B. Jones, et al. An Evaluation of Scalable Application-level Multicast Built Using P2P Overlays. In *Proc. of Infocom*, 2003.
[5] R. Chandra, V. Ramasubramanian, et al. Anonymous Gossip: Improving Multicast Reliability in Ad-Hoc Networks. In *Proc. of ICDCS*, 2001.
[6] R. Flury and R. Wattenhofer. Routing, Anycast, and Multicast for Mesh and Sensor Networks. In *Proc. of Infocom*, 2007.
[7] Q. Huang, C. Lu, et al. Spatiotemporal multicast in sensor networks. In *Proc. of SenSys*, 205–217, 2003.
[8] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proc. of MobiCom*, 2000.
[9] S. R. Madden, M. J. Franklin, et al. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. on Database Sys.*, 30(1):122–173, 2005.
[10] A. Mainwaring, J. Polastre, et al. Wireless Sensor Networks for Habitat Monitoring. In *Proc. of WSNA 2002*, 88–97, 2002.
[11] S. Ratnasamy, D. Estrin, et al. Data-Centric Storage in Sensornets. In *Proc. of SIGCOMM*, 2002.
[12] S. Ratnasamy, P. Francis, et al. A scalable content-addressable network. In *Proc. of SIGCOMM*, 161–172, 2001.
[13] P. Rentala, R. Musunuri, et al. Survey on sensor networks. In *Proc. of MobiCom*, 2001.
[14] SeNSim. `http://sensim.gforge.inria.fr/`, ASAP Research Project, INRIA Rennes, FR, 2005-2008.
[15] N. Thepvilojanapong, Y. Tobe, et al. HAR: Hierarchy-based Anycast Routing Protocol for Wireless Sensor Networks. In *Proc. of SAINT*, 204–212, 2005.
[16] J. E. Wieselthier, G. D. Nguyen, et al. Energy-efficient broadcast and multicast trees in wireless networks. *Mobile Networks and Applications*, 7(6):481–492, 2002.